I. Abstract

The goal was to implement the double-differenced CDGPS algorithm while calculating a precise estimate of double-differenced beat carrier phase ambiguities as integers for the determination of the position of various surveyed locations. This particular project is an excellent extension of the GPS surveying techniques taught in class that allows for learning a new method that is both challenging and intriguing: a real application of the coursework that goes beyond our current grasp of GPS. The project can be broken down into a few major pieces. Collecting the proper data, calculating the bias (real-value then integer-ambiguity), and then finally calculating the position of the static/dynamic data using the best possible bias value to show that the CDGPS technique reproduces the same locations as the pre-surveyed positions. To solve for all these pieces, we used MATLAB functions that implemented the equations. The functions ran in a script to complete the steps outlined above. A key point to note is that the GPS time of each sample chosen has to be the same for all receivers. Collecting data consisted of gathering the ephemerides and atmospheric data describing the satellites and observational data and L1 beat carrier phases for the receivers. Formatting the data to extract the proper satellites, times, and other important pieces of data like pseudoranges and beat carrier phases, we then used the information available to calculate values for various intermediary parts of the main functions and equations:

Linearized form of Double-Differenced Beat Carrier Phase Equation-

$$l_k = A_k x_k + \lambda_{L1} \nabla\Delta\beta$$

Cost Function-

$$\tilde{J}_{LSQ}(\nabla\Delta\beta) = \frac{1}{2}\sum_{k=1}^{M}(l_k - \lambda_{L1}\nabla\Delta\beta)^T \tilde{Q}_k^{-1}(l_k - \lambda_{L1}\nabla\Delta\beta)$$

Real-Valued Bias Solution (which is just minimizing the above cost function)-

$$\nabla\Delta\beta_r = \frac{1}{\lambda_{L1}}\left(\sum_{k=1}^{M}\tilde{Q}_k^{-1}\right)^{-1}\left(\sum_{k=1}^{M}\tilde{Q}_k^{-1} l_k\right)$$

So solving for the intermediary but very significant values *l* and *A* matrices as well as the Q_{tilda} matrix, we could solve for the real-valued bias. This particular value isn't accurate enough though so using something known as a "hyper rectangle" we found integer-ambiguity lattice points which are integer values for the biases. Through a series of trial and error where we once again tried to minimize the cost function but with the integer values, we determined the best possible bias values to use. With the calculated biases and matrix values, the final step was to implement the linearized form of the double-differenced beat carrier phase equation to find x_k. x_k is only the change of position so we iterate through with the Newton-Raphson method to find the where x_k converges to. This entire process is done for every sample to determine the position at each sample time.

From our results, we can definitively say that the CDGPS method of determining position is incredibly accurate but not the most efficient way. Run time is a bit too slow to run practically in real world situations and it requires a large amount of data. But in general, when combined with sufficient and accurate data, CDGPS can calculate position with error that is consistently at most within a few meters. The error is nearly all in the vertical direction and can be potentially attributed to imperfect surveying or the fact that our time samples are not precisely equal (off by 0.03 seconds).

II. Introduction

The primary purpose here was to demonstrate the double-differenced CDGPS algorithm by testing the technique we coded completely in MATLAB with real life data gathered from static and dynamic sources. In short, we used two receivers and differenced their many elements to determine position. All the code can be seen in the appendix with clear comments showing each individual file's function. The overall structure of the project has the overall loop for the Newton-Raphson approximation and the entire code base. Before the Newton-Raphson loop though, the data gathered from the receivers are loaded and formatted for what we need. An initial loop within the overall Newton-Raphson cycles through all the samples to obtain the l_k matrix, A_matrix, and Qtilda values. Global variables held the sum of the Qtildas and matrices held the l_k and A values at each sample time. The real-valued bias is then calculated with this information. To update the bias, we call hyper rectangle and then loop through the result of that function to find the best bias by entering it into the cost function. Finally using that bias we finally use the double-differenced equation to calculate for Δx,y,z and update it from our initial loop. This is all documented in the code located in the appendix. Though we can break down the process into such a few steps, it is much more involved technique that truly applies all the information taught in the course.

In order to get the best data, we had functions scour our data to make sure that the time samples were as close as possible and that the satellites were visible for both receivers over the entire sample that we took.
As a piece of user friendliness and ease of displaying the data we desire, we also have options written into the script that allows for easy choosing of which data sets to use as well as concise outputs of the final result in graphical and textual form. Graphs of vertical, north, and east error and average position across our entire static sample are all displayed. To accomplish this, we store the final positions into a matrix and calculate and plot the data within the matrix.

III. Implementation of the CDGPS

To fully implement the CDGPS method, we had to test it with various sets of data. The ultimate goal of the CDGPS is to calculate a Δx,y,z value to add to our initial guess. To get the most accurate value for $x_k = \begin{bmatrix} X_a(T_{Rak}) - X_{a0}(T_{Rak}) \\ Y_a(T_{Rak}) - Y_{a0}(T_{Rak}) \\ Z_a(T_{Rak}) - Z_{a0}(T_{Rak}) \end{bmatrix}$ we used the Newton-Raphson method to iterate through and update the value of x_k until it converged to a value. This was usually obtainable within a few iterations of the Newton-Raphson method. This process was repeated for every sample we had where the two GPS times of the receivers and satellites used for calculations were the same. This was repeated for every sample for multiple reasons. It allows for many tests of the CDGPS and also allowed for an average across all sample times to achieve a more comprehensive result. Instead of just having results for one specific time, an entire set of results can help eliminate potential outliers. The repetition over every sample time is also convenient for dynamic receivers where the position changes every second like in a moving vehicle. In such a scenario, sampling across the entire data set is necessary to really map out the receiver path. So we calculated the position of the receivers across the entire data set by calculating a value for x_k and iterating through with the Newton-Raphson method to get the best answer. Calculating x_k is the bulk of this project and the CDGPS though.

To calculate x_k we had to implement the linearized form of the double-differenced carrier phase equation which is stated above in Part I, Abstract. Implementing the equation is very straight forward simply substituting each variable with its respective value. As stated in the course text,

$$l_k = \begin{bmatrix} \{\lambda_{L1}\nabla\Delta\phi_{abk}^{12} - (1+\dot{\delta}_k^1)(\rho_{ak0}^1 - \rho_{bk}^1) + (1+\dot{\delta}_k^2)(\rho_{ak0}^2 - \rho_{bk}^2) + c(\dot{\delta}_k^1 + \dot{\delta}_k^2)\Delta T_{Rabk}\} \\ \{\lambda_{L1}\nabla\Delta\phi_{abk}^{13} - (1+\dot{\delta}_k^1)(\rho_{ak0}^1 - \rho_{bk}^1) + (1+\dot{\delta}_k^3)(\rho_{ak0}^3 - \rho_{bk}^3) + c(\dot{\delta}_k^1 + \dot{\delta}_k^3)\Delta T_{Rabk}\} \\ \vdots \\ .\{\lambda_{L1}\nabla\Delta\phi_{abk}^{1N} - (1+\dot{\delta}_k^1)(\rho_{ak0}^1 - \rho_{bk}^1) + (1+\dot{\delta}_k^N)(\rho_{ak0}^N - \rho_{bk}^N) + c(\dot{\delta}_k^1 + \dot{\delta}_k^N)\Delta T_{Rabk}\} \end{bmatrix}$$

*Note, the equation for *A_k* can be viewed in the appendix and in the code itself

All the variables within these matrices can be grabbed from observed data from the receivers.
λ_{L1} is a constant determined by the L1 signal wavelength.
ϕ, the beat carrier phase is outputted by the receiver when collecting data. In MATLAB it can be read in as a .mat file. The other components require a bit of calculation but the components of the calculation are all within the collected data. The range, ρ, can be calculated by first using the pseudo range obtained by the receiver and determining the satellite position and then using the distance formula to find the range between the satellite and the receiver position. For the known receiver, the receiver position will just simply be the recorded receiver position and for the unknown, the receiver position would just be the current position recorded from the Newton-Raphson method. The real time difference between receivers A and B, ΔT, is simply the GPS time observed subtracted by δ_{rec0}, the receiver clock error. Find the real times for receiver A and receiver B and the difference between the two to determine ΔT. And finally, the clock offset dot factor can be derived from the equation for clock error but taking the derivative:

$$\dot{\delta} = a_{f1} + 2a_{f2}\Delta T + \left(-2\sqrt{\frac{\mu_E}{c^2}}\right)(e\sqrt{a})\cos(E)\cdot\dot{E}$$

The variables here are all within the ephemerides file and can be pulled out based on the satellite. As for the partial derivatives of range with respect to X, Y, and Z, that is a simple calculation of the difference between the receiver and satellite X, Y, and Z values and dividing by the range.
l_k and A_k will both be used extensively in the future so we store each of these values for each time sample in a matrix (a matrix for each l and A) that's size [number of samples] by 1 where the position in the matrix represents the l or A value at that sample. Simultaneously, within the same loop, we also calculate Qtilda within the *calculateQtilda.m* function that also returns the A matrix. This makes the most sense because we will need the sum of Qtilda across all samples in order to solve for the bias and to solve for Qtilda, we need the A matrix.

$$\tilde{Q}_k = \left[Q_{DD}^{-1} - Q_{DD}^{-1}A_k\left(A_k^T Q_{DD}^{-1}A_k\right)^{-1}A_k^T Q_{DD}^{-1}\right]^{-1}$$

We don't actually take the inverse here though.

The A matrix here is already calculated alongside Qtilda. Qdd is a simple matrix filled with constants also known as the symmetric weighing matrix.

$$Q_{DD} = \begin{bmatrix} 4 & 2 & 2 & \cdots & 2 \\ 2 & 4 & 2 & \cdots & 2 \\ 2 & 2 & 4 & \cdots & 2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2 & 2 & 2 & \cdots & 4 \end{bmatrix}$$

We calculate *Qtilda* for each sample and per cycle of the loop, add it to a global variable that keeps a running total. The same is done for *Qtilda*l_k* for both of these will be used in the future to calculate bias.

This part of the CDGPS process is very straightforward and we implemented it within separate functions to calculate each matrix (Appendix, *l_kmatrix.m, calculateQtilda.m*). To calculate each portion we used a revised version of the code provided throughout the course by Professor Psiaki. Looking at *solveposod_n.m* we used the original framework of *solveposod.m* but did not iterate through to find the position. We only calculated the position without iterating because our script already iterates for the Newton-Raphson; iterating twice would be redundant and the effects are shown in our results. Using this particular method changed our error from around one meter in the eastwardly direction to only a few centimeters.

So far, we have been setting up the framework to calculate the real-valued bias and final solution of the linearized double-differenced beat carrier phase equation. The matrices of *l_k, A*, and the running totals of *Qtilda* and *Qtilda*l_k* are all tools to maximize efficiency of the calculation of bias and *x_k*.
The initial real-value calculation of the bias can be calculated from the real-value bias solution equation stated in the abstract. We have both summations calculated already in our first step so obtaining the real-valued bias is very simple. Something we did to increase accuracy and efficiency was to not actually take the inverse of the Qtilda matrix as you are supposed to in the equation. Since we will take the inverse of Qtilda to calculate the bias, we realized it would be excessive to take the inverse of an inverted matrix and that it would be more efficient to simply keep Qtilda the same instead of having to take the inverse twice.

The complex part is determining the integer values for the biases. This requires a technique involving the 'hyper rectangle' that was mentioned many times before. The basic idea is that the real-valued biases are the where the hyper rectangle is centered. The halfwidths of its sides are defined by a theoretical standard deviations of the real-valued biases. The integers within or on this hyperrectangle are the values we are looking to find.
In our code, we needed very little to implement the technique. Code for the hyper rectangle was already provided and we just needed to calculate the inputs and understand what to do with the outputs. The inputs are the real-valued biases that we have already calculated, the theoretical standard deviations to the real-valued biases, and an aeta value which is the number of standard deviations that will be used to define the half width of the hyper rectangle.

All these values can be calculated and quite intuitively. The theoretical standard deviation is a bit convoluted but it derived from the calculation of the variance of the beat carrier phase. From the course text, we can say that the error covariance in the real-valued estimate of the bias:
$E\{(d\nabla\Delta\beta_r)(d\nabla\Delta\beta_r)^T\} = \sigma_\phi^2\left(\sum_{k=1}^{M}\tilde{Q}_k^{-1}\right)^{-1}$
The square roots of the diagonal elements of this covariance matrix are the theoretical standard deviations. In this case it is important to note that we will be assuming that the standard deviation of the beat carrier phase is 1. Here you can see why we had Qtilda sums created earlier, it is used throughout the entire code when calculating bias.
To determine the value for aeta, the idea is once again to minimize the cost function in some form. Here we want the least-squares cost. This can be simplified down into this equation:

$$\eta = \left\{2\left[\tilde{J}_{LSQ}(\nabla\Delta\beta) - \tilde{J}_{LSQ}(\nabla\Delta\beta_r)\right]/\sigma_\phi^2\right\}^{0.5}$$

Inputting these two values along with the real-valued biases into *hyperrectangle.m* returns all the integer-ambiguity lattice points. Next we have to determine which set to use. This is done via another loop that loops through all the columns and finds the vector that has the smallest cost (this is calculated via the cost function).

We now have all the components necessary to implement the linearized double differenced beat carrier phase equation. We pulled this out as a separate function from our script and had the inputs be the known values and the output be the unknown: *x_k*. This is iterated through until the value converges and then combine it with the observed position (our guess) to achieve our final answer. This is done for every sample and the result is saved in a huge matrix for us to come up with meaningful conclusions and results.
There is always a huge amount of testing that is required and we had a significant amount of data to test our code with and with very promising results. Below is the information we gathered from our own implantation of the CDGPS technique.

A. Results: All data was taken the night of December 3, 2012 starting at roughly 7:30PM EST
First, it is unfortunate but we could not get any strong dynamic data. We have two sets of excellent static data but the mobile receivers frequently lost satellite signal no matter how much we drove around rendering most of the data collected useless because we couldn't get a solid number of satellites to use and frequently lost our position for any form of calculation whether it be via pseudoranges or CDGPS.

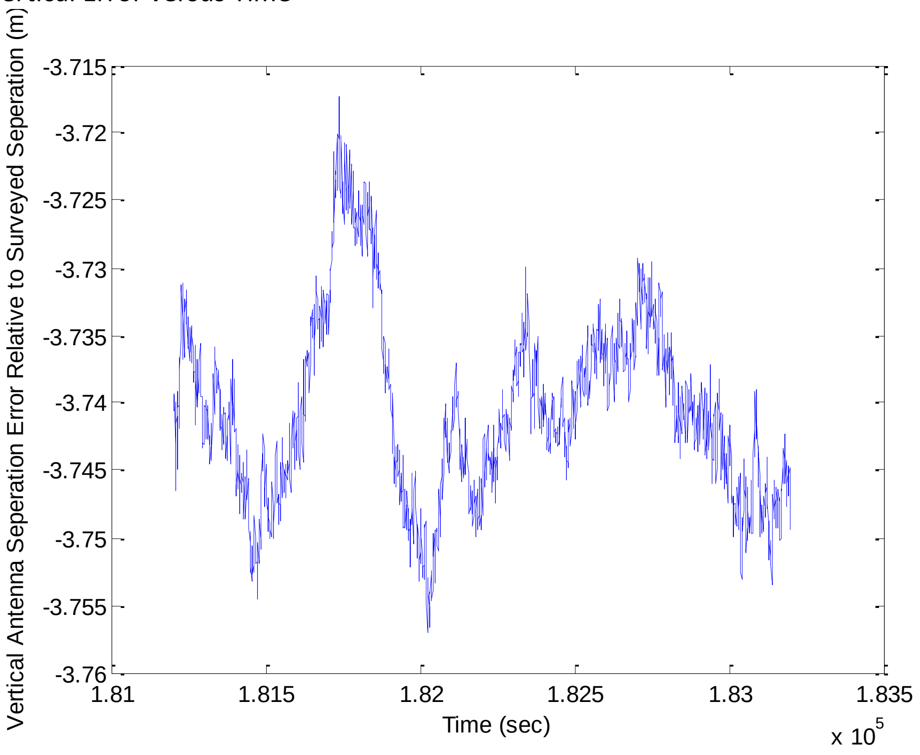
Comparing the vertical error to time graph to the example results in the course textbook, the vertical component matches the values found the book nearly exactly. The error of 3 some meters is a little larger than what we would expect but this can be attributed to many different reasons ranging from imperfect surveying to begin with, inaccurate atmospheric conditions recorded, or having receiver times not being perfectly aligned. The error is nearly all in the vertical direction with very little in the north/east direction (or lat/long, either way, it's mostly vertical error).

The results for the north and east error graphs are the most interesting. They are drastically different than the results that were expected from the course text and we don't have an explanation for it. The error in the northern direction remains very constant but it changes in the eastern direction. In the second set of data seen below, the north error actually remains at nearly exactly 0 the entire time but the eastern direction still has a much wider range. Some ideas we had about this included the fact that we do this based on a relative basis. Perhaps the results would be different if we took the results but had the unknown receiver flipped. Some other observations we made was the large amount of time it took for us to run the code. We attempted to optimize it as best as possible but it still took quite a few minutes to run. Originally the code took at least 10 minutes to run but at the end we pushed it to a few minutes. There was major improvement but still not enough to be usable in real life.

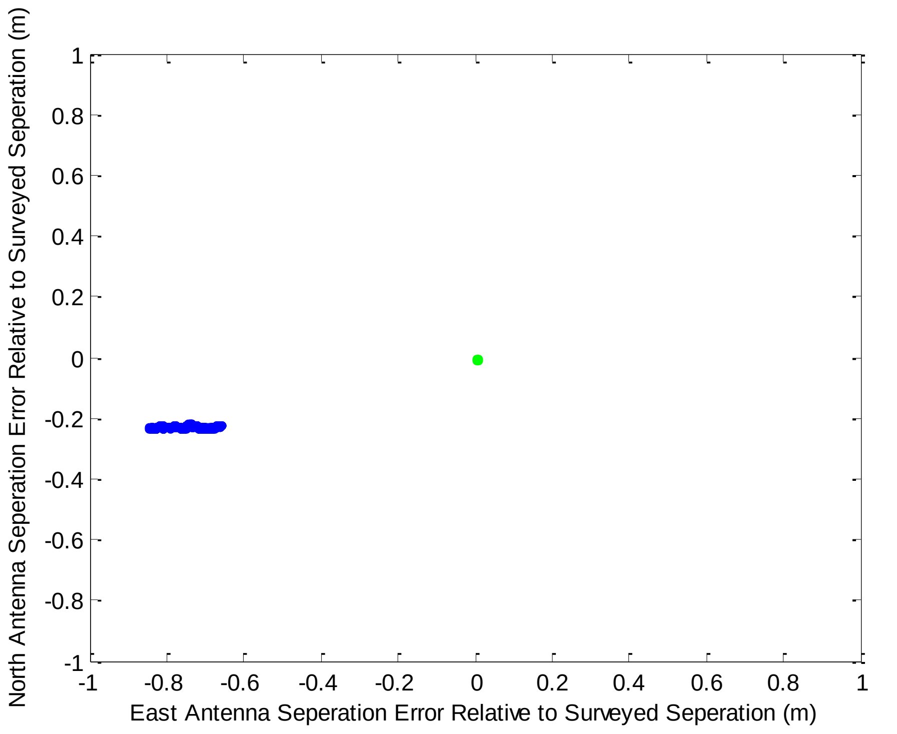
All the graphs are shown below with information on which receivers were used and other pertinent data.

Static Results: (the two receivers we had
Rhodes Hall North and Rhodes Hall South with Rhodes Hall North being the static receiver.

The surveyed position of the unknown receiver is: 42.443545° -76.481790° 253.940000m
The average DDGPS solution of the unknown receiver is: 42.443543° -76.481799° 250.199537m
The norm error between the surveyed solution and the DDGPS solution is: 3.821196m
Vertical Error versus Time

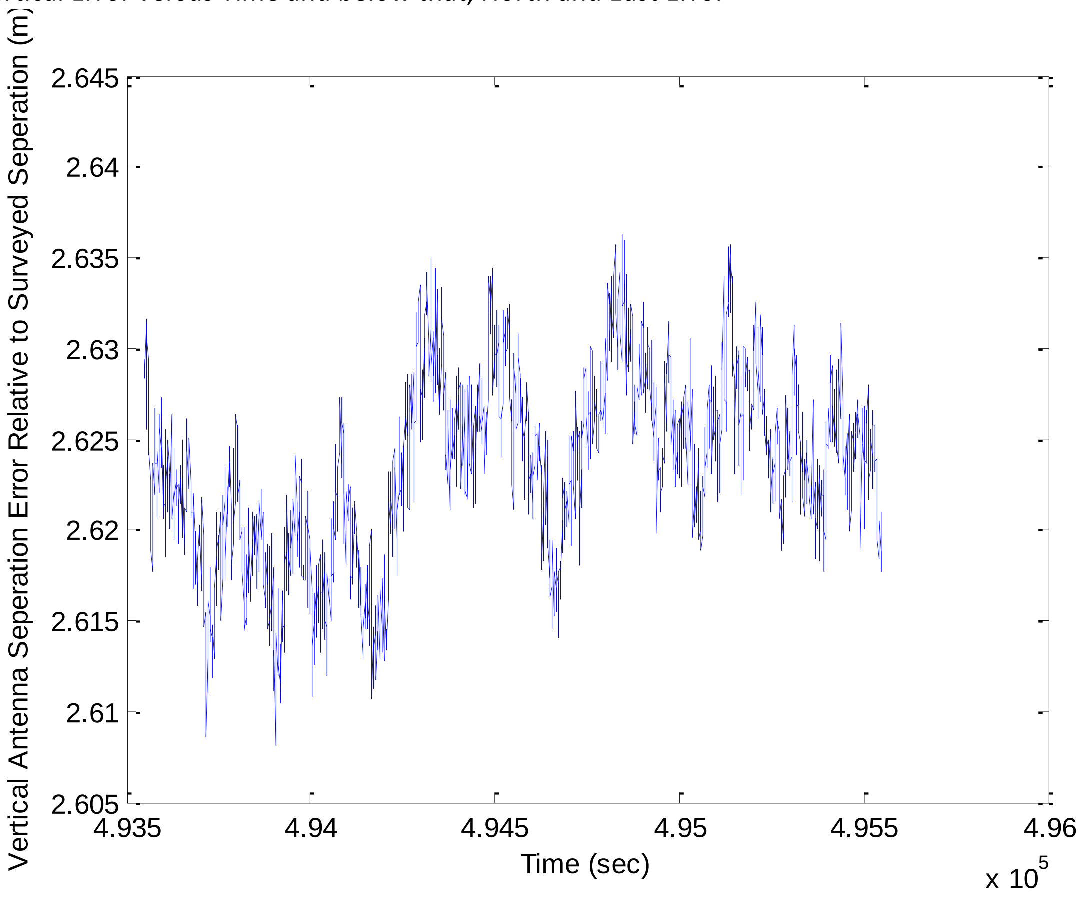


North and East Error

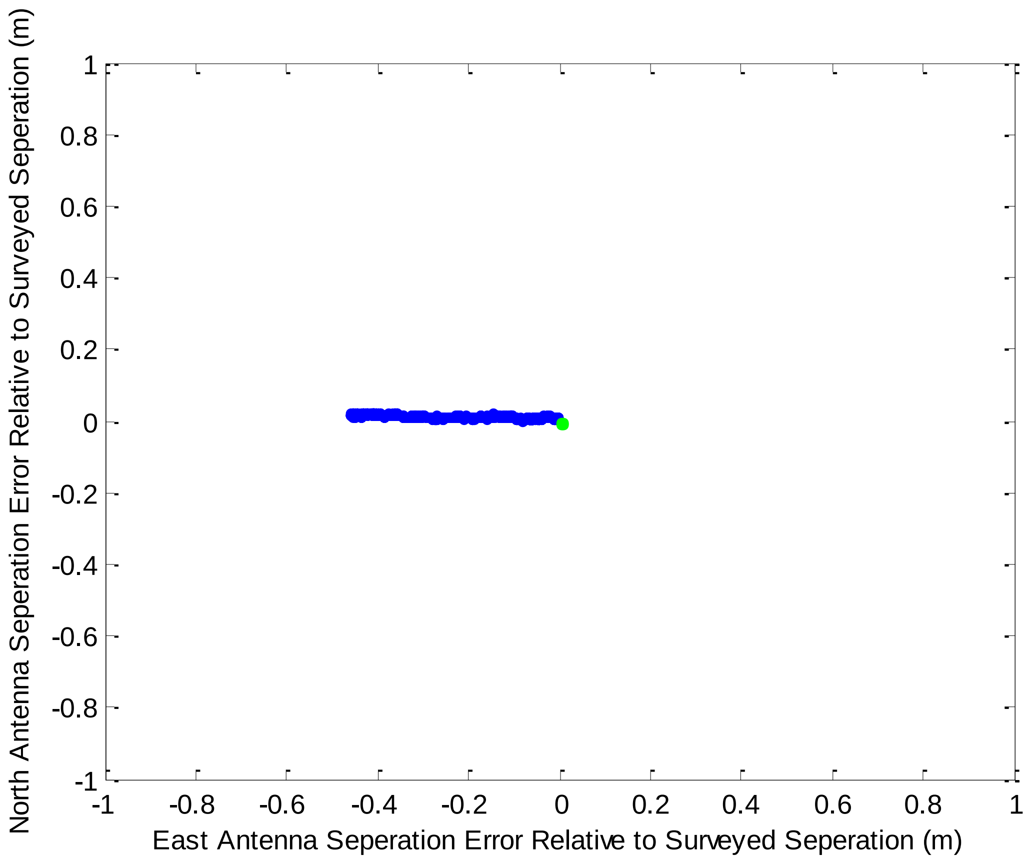


Upson Hall and Rhodes Hall South with Upson Hall being the static receiver
The surveyed position of the unknown receiver is: 42.443327° -76.481433° 250.335000m
The average DDGPS solution of the unknown receiver is: 42.443327° -76.481435° 252.958461m
The norm error between the surveyed solution and the DDGPS solution is: 2.631307m

Vertical Error versus Time and below that, North and East Error



The graphs above and below are the results of the same CDGPS technique used in the first set of data. Both are static receivers. Compared to the first set of data, this set is a little more accurate but the shape of the error remains the same: an oscillating error for the vertical direction (most of the error is still in the vertical direction) and a constant north error. These values change depending on how we iterate to calculate the ranges and the error also is slightly unexpected compared to the course notes. These are all potential parts of CDGPS that we could look into.



IV. Summary

The CDGPS provides a method in determining receiver location that has been far more accurate than any other technique we've explored so far. It is a very formulaic approach that is primarily based off the fundamental double-differenced beat carrier phase equation but linearizing it so it can be expressed in a much more condensed form that works well with matrices. Using previous knowledge on how to define and acquire data like pseudo range, real time, and the beat carrier phase we utilized all that previous knowledge to apply it to double-differencing equations and the calculation of the real-valued and integer ambiguities. We made some assumptions and altered code accordingly and the results show promising but mixed results. This allowed for an opportunity to expand our knowledge to actually test something we've learned and fully implement and test it.

V. Conclusion

CDGPS is certainly a very accurate method to determine receiver location. From our results, it is the most accurate technique we have used so far and the error could very easily be due to faulty surveying or imperfect time samples (the two receiver clock differences were roughly 0.03 seconds which can easily be a huge amount considering the rate at which satellites move). There are also things to consider such as the efficiency of the method (though this could simply be us with our code, our machines, and not having the full knowledge of GPS) as well as limitations such as the necessary for proximity of receivers, shared satellites, and a very close time sampling. Despite its limitations, it does show impressive results and can be used in many scenarios.

VI. Recommendations

Purely based off our results, it seems like a great tool to use provided the known receivers are located at a close enough proximity. It would perfect to use for static receivers in a confined space but would not be optimal with dynamic data as already seen by our attempts to find enough matching satellites and coordinate close enough GPS times. This would also work better with static receivers or data not in real time because it takes quite a bit of time to calculate everything. Though only a few minutes at most, that can't be used for real time/moving vehicles. Imagine a fighter jet receiving data about GPS locations with a minute delay. So based off our results, CDGPS is indeed very accurate but because of the precise data it needs and valuable time to calculate final position, it works best for static receivers in a confined space rather than moving data because of its efficiency (or our efficiency) and need for close known receivers.

VII. References

Course Text and various MATLAB codes. All the equations and CDGPS techniques used are from the course text which were all accessed at:
Psiaki, Mark. (2012). MAE4150/ECE4150 GPS: Theory and Design Fall 2012. December 10, 2012, http://web.mae.cornell.edu/courses/MAE4150.

VII. Appendix

MATLAB code as well as other equations to take note of:

$$\rho_{ak0}^{i} = \sqrt{\left[X_{a0}(T_{Rak}) - X^{i}(T_{ak}^{i})\right]^2 + \left[Y_{a0}(T_{Rak}) - Y^{i}(T_{ak}^{i})\right]^2 + \left[Z_{a0}(T_{Rak}) - Z^{i}(T_{ak}^{i})\right]^2}$$

$$\frac{\partial \rho_{ak}^{i}}{\partial X_a} = \frac{X_{a0}(T_{Rak}) - X^{i}(T_{ak}^{i})}{\rho_{ak0}^{i}} \qquad \frac{\partial \rho_{ak}^{i}}{\partial Y_a} = \frac{Y_{a0}(T_{Rak}) - Y^{i}(T_{ak}^{i})}{\rho_{ak0}^{i}} \qquad \frac{\partial \rho_{ak}^{i}}{\partial Z_a} = \frac{Z_{a0}(T_{Rak}) - Z^{i}(T_{ak}^{i})}{\rho_{ak0}^{i}}$$

$$\nabla\Delta\beta = \begin{bmatrix} \nabla\Delta\beta_{ab}^{12} \\ \nabla\Delta\beta_{ab}^{13} \\ \nabla\Delta\beta_{ab}^{14} \\ \vdots \\ \nabla\Delta\beta_{ab}^{1N} \end{bmatrix}$$

$$A_k = \begin{bmatrix} \left\{(1+\dot{\delta}_k^1)\frac{\partial \rho_{ak}^1}{\partial X_a} - (1+\dot{\delta}_k^2)\frac{\partial \rho_{ak}^2}{\partial X_a}\right\} & \left\{(1+\dot{\delta}_k^1)\frac{\partial \rho_{ak}^1}{\partial Y_a} - (1+\dot{\delta}_k^2)\frac{\partial \rho_{ak}^2}{\partial Y_a}\right\} & \left\{(1+\dot{\delta}_k^1)\frac{\partial \rho_{ak}^1}{\partial Z_a} - (1+\dot{\delta}_k^2)\frac{\partial \rho_{ak}^2}{\partial Z_a}\right\} \\ \left\{(1+\dot{\delta}_k^1)\frac{\partial \rho_{ak}^1}{\partial X_a} - (1+\dot{\delta}_k^3)\frac{\partial \rho_{ak}^3}{\partial X_a}\right\} & \left\{(1+\dot{\delta}_k^1)\frac{\partial \rho_{ak}^1}{\partial Y_a} - (1+\dot{\delta}_k^3)\frac{\partial \rho_{ak}^3}{\partial Y_a}\right\} & \left\{(1+\dot{\delta}_k^1)\frac{\partial \rho_{ak}^1}{\partial Z_a} - (1+\dot{\delta}_k^3)\frac{\partial \rho_{ak}^3}{\partial Z_a}\right\} \\ \vdots & \vdots & \vdots \\ \left\{(1+\dot{\delta}_k^1)\frac{\partial \rho_{ak}^1}{\partial X_a} - (1+\dot{\delta}_k^N)\frac{\partial \rho_{ak}^N}{\partial X_a}\right\} & \left\{(1+\dot{\delta}_k^1)\frac{\partial \rho_{ak}^1}{\partial Y_a} - (1+\dot{\delta}_k^N)\frac{\partial \rho_{ak}^N}{\partial Y_a}\right\} & \left\{(1+\dot{\delta}_k^1)\frac{\partial \rho_{ak}^1}{\partial Z_a} - (1+\dot{\delta}_k^N)\frac{\partial \rho_{ak}^N}{\partial Z_a}\right\} \end{bmatrix}$$